APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: USE OF A FUTURE FILE FOR DATA ADDRESS CALCULATIONS IN A PIPELINED PROCESSOR

APPLICANT: WILLIAM C. ANDERSON AND RYO INOUE

USE OF A FUTURE FILE FOR DATA ADDRESS CALCULATIONS IN A PIPELINED PROCESSOR

BACKGROUND

[0001] A pipelined processor, such as a microprocessor for a personal computer or a digital signal processor, may include a data address generator (DAG) for generating speculative memory addresses for data which is transferred between memory and registers in the processor. In generating a data address, the DAG may update data address values used to calculate the data addresses. The updated data address values travel down the pipeline until they reach a write back (WB) stage, at which point they may be committed to an architectural file.

[0002] The DAG may use previously updated data address values that are still in the pipeline to generate a new data address. Some architectures include forwarding paths in each stage between the DAG, in an address calculation (AC) stage and the WB stage and utilize detection and multiplexing structures to locate and access the required updated values in the pipeline 102. Some problems associated with this type of architecture stem from the complexity of the logic required to detect data dependencies and to forward the appropriate operands to the

execution units and increased power consumption due to the additional logic.

BRIEF DESCRIPTION OF THE DRAWINGS

[0003] Figure 1 is a block diagram of a processor including a future file according to an embodiment.

[0004] Figure 2 is a block diagram of an execution pipeline according to an embodiment.

[0005] Figure 3 is a flowchart describing an update operation according to an embodiment.

[0006] Figure 4 is a block diagram of a mobile video unit including a processor according to an embodiment.

DETAILED DESCRIPTION

[0007] Figure 1 is a block diagram illustrating a pipelined programmable processor 100, which includes a future file 102 for storing speculative data address values according to an embodiment. The processor 100 may include an execution pipeline 102 and a control unit 104. The control unit 104 may control the flow of instructions and data through the pipeline 102 in accordance with a system clock 106. During the processing of an instruction, the control unit 104 may direct the various components of the pipeline to decode the instruction and to perform the

corresponding operation including, for example, writing results back to memory or registers.

[0008] Instructions may be loaded into a first stage of the pipeline 102 and be processed through subsequent stages. A stage may process concurrently with the other stages. Data may be passed between the stages in the pipeline 102 in accordance with the system clock signal. Instruction results may emerge at the end of the pipeline 102 in succession.

[0009] The pipeline 102 may include, for example, five stages: instruction fetch (IF), instruction decode (DEC), address calculation (AC), execute (EX), and write back (WB), as shown in Figure 2. According to alternate embodiments, these stages may include sub-stages, e.g., the EX stage may include multiple sub-stages EX1, EX2, etc.

[0010] Instructions may be fetched from a memory device 110 such as, for example, a main memory or an instruction cache during the IF stage by a fetch unit 120 in a clock cycle. An instruction fetched in a clock cycle may be decoded in a subsequent clock cycle in the DEC stage by an instruction decode unit 122. The results may be passed to the AC stage, where a data address generator (DAG) 124 may calculate memory addresses for performing the operation. During the EX stage, an execution unit 126 may perform a

specified operation such as, for example, adding or multiplying two numbers. The execution unit 126 may contain specialized hardware for performing the operations including, for example, one or more arithmetic logic units (ALUs), multiply and accumulate (MAC) units, and barrel shifters. A variety of data may be applied to the execution unit 126 such as the addresses generated by the DAG 124, data retrieved from memory or data retrieved from data registers. During the WB stage, the results may be written back to memory 110 or architectural registers 130.

[0011] The DAG 124 may generate speculative memory addresses for data which is transferred between memory and registers in the processor 100. The DAG 124 may employ different techniques to generate the data addresses, which include, for example, automatic incrementing and circular buffering. The DAG may operate on several values to calculate data addresses. These values may include the following: an index (I) value, which represents an address of data to be accessed; a length (L) value, which represents a length and may be used in circular buffering; a base (B) value, which represents a base address; and a modify (M) value, which may be used to post-modify an index value. The DAG 124 may update the index values in the AC stage when it generates a new data address, and may modify

the other values when performing other DAG functions. The updated values exiting the AC stage may be latched in subsequent stage latches 202 (Figure 2) as they flow down the pipeline 102 and may be written to an architectural file 204 upon reaching the WB stage. The architectural file 204 may include four sets of I, L, B, and M registers to store the updated values.

[0012] Generation of a data address may depend on a previously calculated address. In order to generate the data address, the DAG 124 may need to access an updated I, L, B, or M value that it calculated or modified in a previous cycle. However, this updated value may still be in the pipeline 102, and not yet committed to the architectural file 204.

[0013] In an embodiment, a future file 206 may be provided in the DEC stage. The future file acts as a working file and provides updated I, L, B, and M values to the DAG 124, located downstream in the AC stage. Like the architectural file 204, the future file 206 may include four sets each of I, L, B, and M registers.

[0014] Figure 3 is a flowchart illustrating an update operation 300 according to an embodiment. The flow of the operation described in Figure 3 is exemplary, and blocks in the flowchart may be skipped or performed in different

order and still produce desirable results. Updated values calculated in the AC stage in a clock cycle (block 302) exit the AC stage in the next cycle and are forwarded to the future file 206 via an update bus 210. During normal program flow, the future file is updated each cycle (block 304) to contain the last updated, i.e., most current, copies of the I, L, B, and M values calculated by the DAG 124. These copies are made available to the DAG 124 for subsequent calculations as the updated values travel down the pipeline 102.

[0015] The updated values stored in the future file 206 are speculative until their counterparts in the pipeline are committed to the architectural file 204. A speculative value may be rendered invalid if the instruction to which it corresponds is cancelled (i.e., "killed") (block 306). An instruction may be cancelled, and all writes turned off for that instruction, if it is no longer valid for the current program flow. This may occur, for example, when an interrupt is taken. When an interrupt occurs, all instructions in the pipeline 102 may be cancelled in response to a control signal from the control unit 104, and instructions from an interrupt service routine (ISR) may be fetched and introduced into the pipeline. The instructions may be cancelled, for example, by placing zeroes into the

pipeline latches of the cancelled instructions. After the interrupt has been handled by the ISR, the program counter (PC), which tracks the program flow, may return to a cancelled instruction to resume the program flow. In other words, the pipeline rolls back to the state it had prior to fetching the cancelled instruction. This prior state is characterized by the committed values stored in the architectural file 204.

[0016] If the corresponding instruction is not cancelled, the updated (speculative) values continue down the pipeline 102 until they reach the WB stage, at which point they are committed to the architectural file 204 (block 308). If the corresponding instruction is cancelled, the processor 100 rolls back up to the prior state, described above, when the PC returns from the ISR. To facilitate this roll back, the values in the architectural file 204 may be copied to the future file 206 (block 310) via a restore bus 212, thereby overwriting the speculative updated values stored in the future file. When the program flow resumes, the future file 206 provides the DAG 124 with values which are valid for the current program flow. The future file may then be updated via the update bus 210 in subsequent clock cycles.

[0017] Such a processor 100 may be used in video camcorders, teleconferencing, PC video cards, and High-Definition Television (HDTV). In addition, the processor 100 may be used in connection with other technologies utilizing digital signal processing such as voice processing used in mobile telephony, speech recognition, and other applications.

[0018] For example, Figure 4 illustrates a mobile video device 400 including a processor 100 according to an embodiment. The mobile video device 400 may be a hand-held device which displays video images produced from an encoded video signal received from an antenna 402 or a digital video storage medium 404, e.g., a digital video disc (DVD) or a memory card. The processor 100 may communicate with a cache memory 406, which may store instructions and data for the processor operations, and other devices, for example, a static random access memory (SRAM) 408.

[0019] The processor 100 may be a microprocessor, a digital signal processor (DSP), a microprocessor controlling a slave DSP, or a processor with a hybrid microprocessor/DSP architecture. The processor 100 may perform various operations on the encoded video signal, including, for example, analog-to-digital conversion, demodulation, filtering, data recovery, and decoding. The

processor 100 may decode the compressed digital video signal according to one of various digital video compression standards such as the MPEG-family of standards and the H.263 standard. The decoded video signal may then be input to a display driver 410 to produce the video image on a display 412.

[0020] A number of embodiments of the invention have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the invention. Accordingly, other embodiments are within the scope of the following claims.